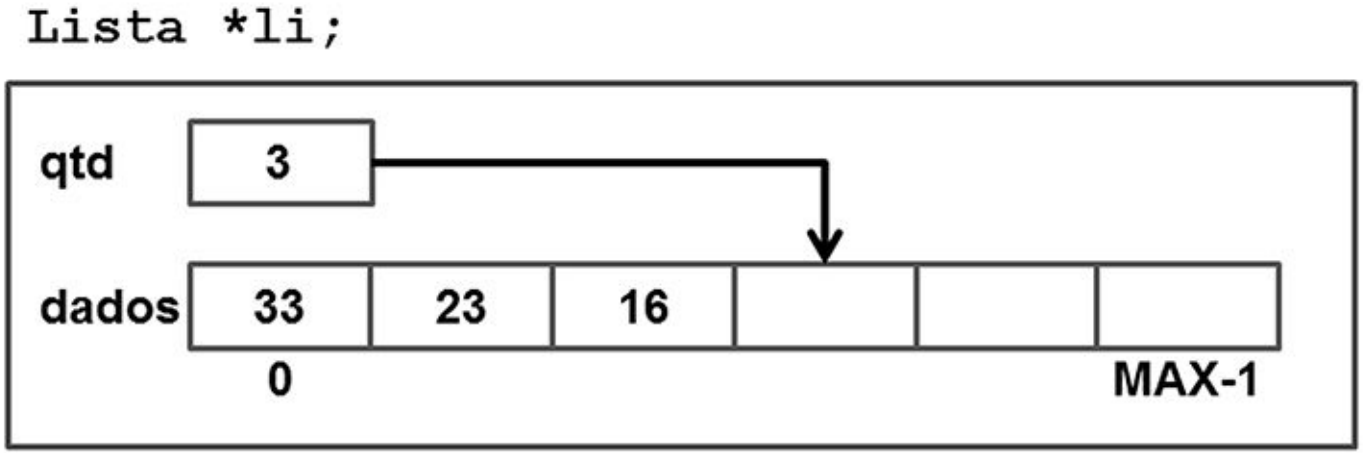
Filas

Sergio Canuto
sergio.canuto@ifg.edu.br

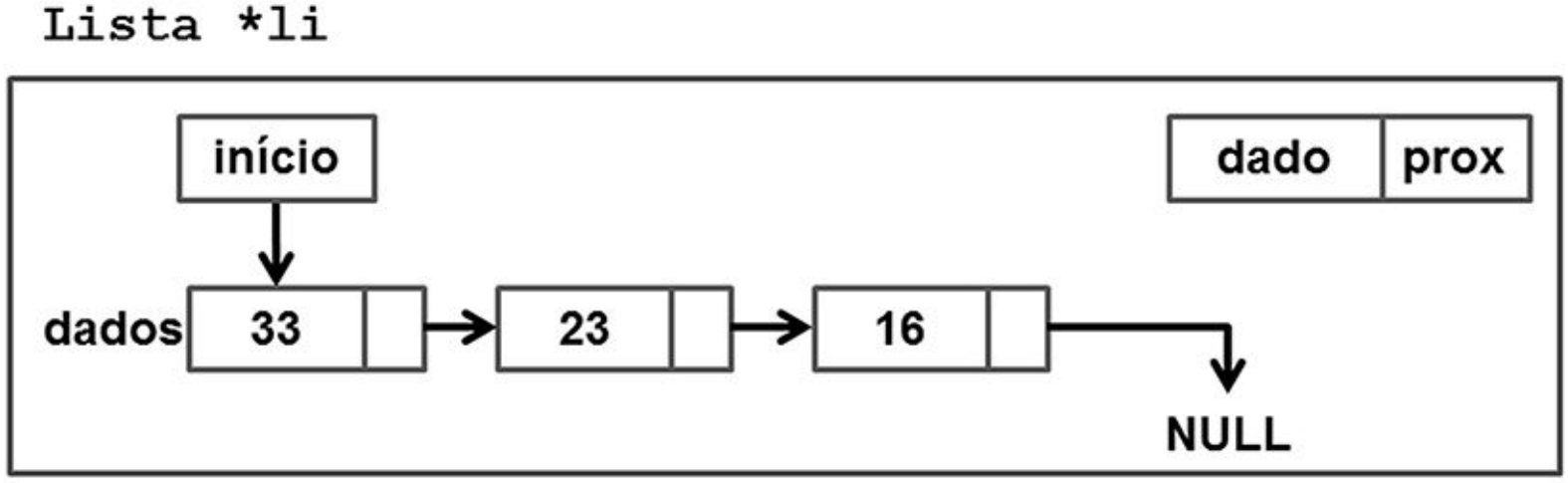
Relembrando...

- **LISTA SEQUENCIAL ESTÁTICA**
  - Uma **lista sequencial estática** ou **lista linear estática** é uma lista definida utilizando alocação estática e acesso sequencial dos elementos



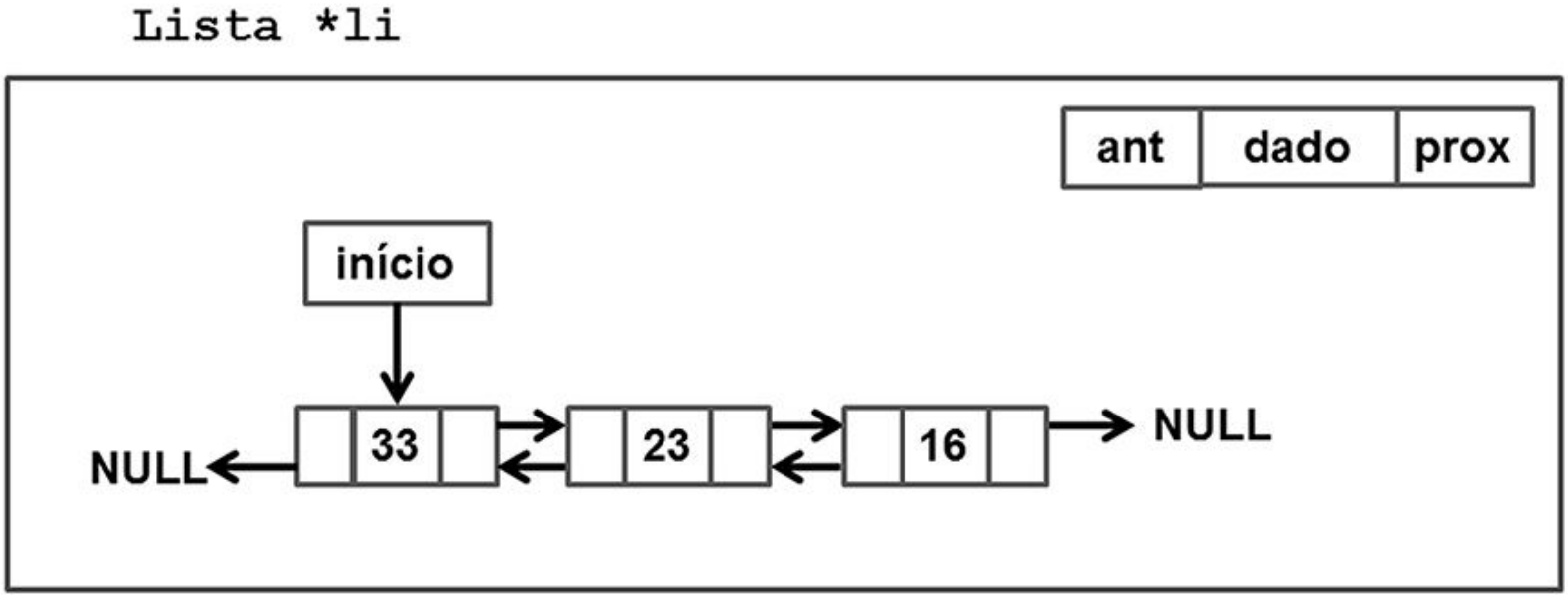
LISTA DINÂMICA ENCADEADA

- Uma **lista dinâmica encadeada** é uma lista definida utilizando alocação dinâmica e acesso encadeado dos elementos. Cada elemento contém:
  - um campo de **dado**, utilizado para armazenar a informação.
  - um campo **prox**, ponteiro que indica o próximo elemento na lista.



LISTA DINÂMICA DUPLAMENTE ENCADEADA

- Diferente da **lista dinâmica encadeada**, esse tipo de lista não possui dois, mas sim **três** campos de informação dentro de cada elemento: os campos **dado**, **prox** e **ant**.
- A presença dos ponteiros **prox** e **ant** garantem que a lista seja encadeada em dois sentidos: no seu sentido normal, aquele usado para percorrer um lista do seu início até o seu final, e no sentido inverso, quando percorremos a lista de volta ao seu início.



Filas - Definição

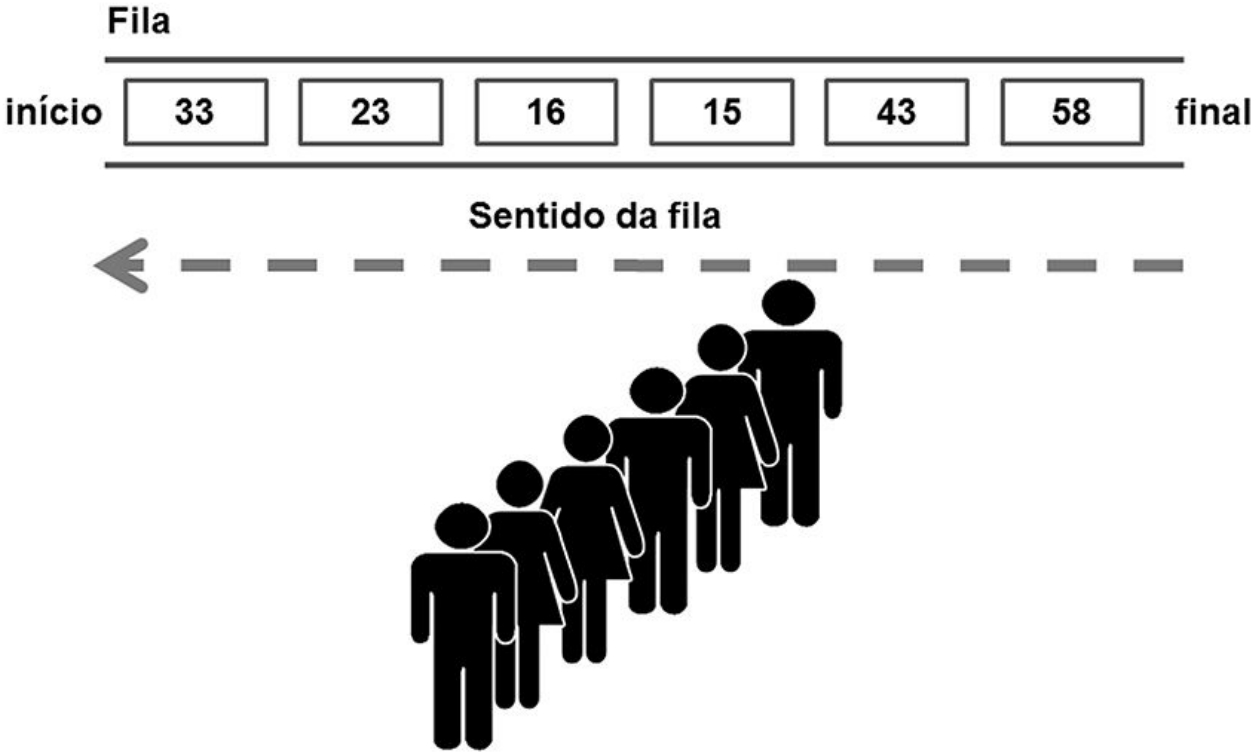
- O conceito de fila, ou *fila de espera*, é algo bastante comum para as pessoas. Afinal, somos obrigados a enfrentar uma fila sempre que vamos ao banco, ao cinema etc.
- Na computação, uma fila nada mais é do que um conjunto finito de itens (de um mesmo tipo) esperando por um serviço ou processamento. Trata-se de um controle de fluxo muito comum na computação.
- Um exemplo bastante comum da aplicação de filas é o gerenciamento de documentos enviados para a impressora.
- As filas são implementadas e se comportam de modo muito similar às listas, sendo, muitas vezes, consideradas um tipo especial de lista em que a inserção e a remoção são realizadas sempre em extremidades distintas.

Filas - Definição

- Neste caso, a inserção de um item é feita de um lado da fila, enquanto a retirada é feita do outro lado.
- Se quisermos acessar determinado elemento da fila, deveremos remover todos os que estiverem à frente dele.
- Por esse motivo, as filas são conhecidas como estruturas do tipo **primeiro a entrar, primeiro a sair** ou FIFO (First In First Out): os elementos são removidos da fila na mesma ordem em que foram inseridos.

Filas - Definição

- **Primeiro a entrar, primeiro a sair** ou FIFO (First In First Out):



Filas - Implementações

- Existem dois tipos de implementações principais para uma fila:
  - **Alocação estática com acesso sequencial**: o espaço de memória é alocado no momento da compilação do programa, ou seja, é necessário definir o número máximo de elementos que a fila irá possuir. Desse modo, os elementos são armazenados de forma consecutiva na memória (como em um array ou vetor) e a posição de um elemento pode ser facilmente obtida a partir do início da fila.
  - **Alocação dinâmica com acesso encadeado**: o espaço de memória é alocado em tempo de execução, ou seja, a fila cresce à medida que novos elementos são armazenados, e diminui à medida que elementos são removidos. Nessa implementação, cada elemento pode estar em uma área distinta da memória, ambas não necessariamente consecutivas. É necessário então que cada elemento da fila armazene, além da sua informação, o endereço de memória onde se encontra o próximo elemento. Para acessar um elemento, é preciso percorrer todos os seus antecessores na fila.

Filas - Implementações

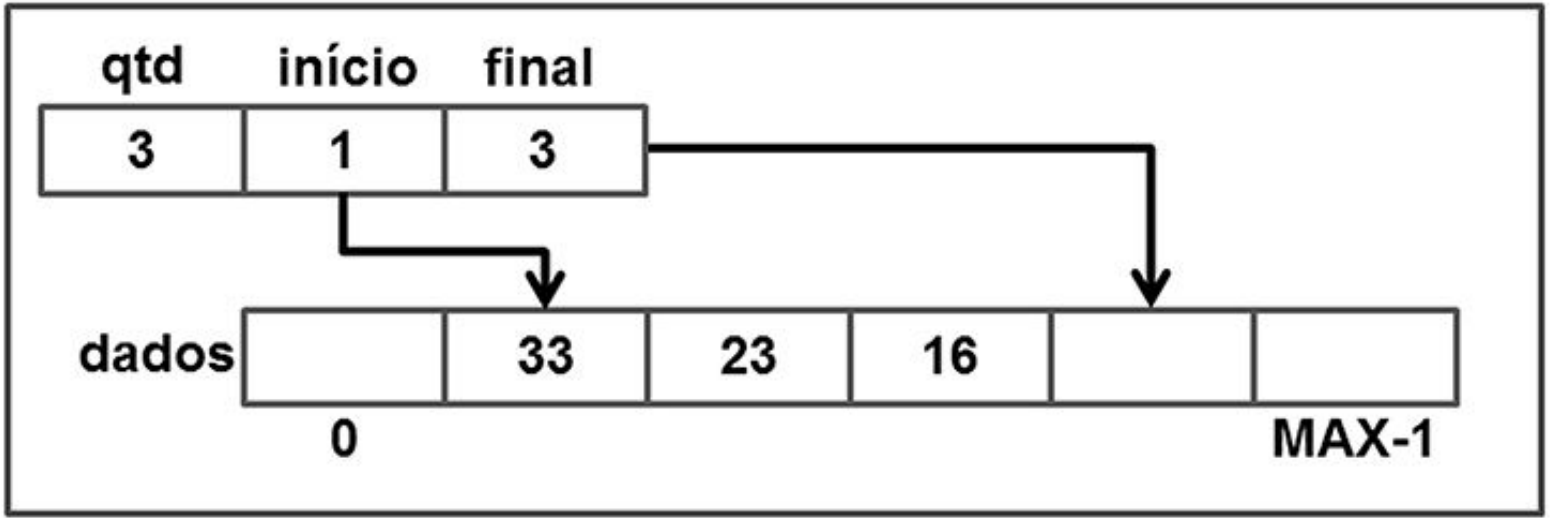
Independentemente do tipo de alocação e acesso usado na implementação de uma fila, as seguintes operações básicas são sempre possíveis:

- Criação da fila.
- Inserção de um elemento no final da fila.
- Remoção de um elemento do início da fila.
- Acesso ao elemento do início da fila.
- Destruição da fila.
- Além de informações com tamanho, se a fila está cheia ou vazia.

Fila - Sequencial Estática

- Uma **fila sequencial estática** é uma fila definida utilizando alocação estática e acesso sequencial dos elementos.
- Além do array, essa fila utiliza três campos adicionais para guardar o **início**, o **final** e a **quantidade** de elementos (**dados**) inseridos na fila.

```
Fila *fi;
```



Fila - Sequencial Estática

- A implementação de uma **fila sequencial estática** é muito parecida com a implementação de uma **lista sequencial estática**.
- Diferenças:
  - Além da informação de quantidade (qtd), o início/final da fila.
  - Fila possui uma regra para inserção e outra para remoção.

Arquivo FilaEstatica.h

```
01  #define MAX 100
02  struct aluno{
03      int matricula;
04      char nome[30];
05      float n1,n2,n3;
06  };
07  typedef struct fila Fila;
08
09  Fila* cria_Fila();
10  void libera_Fila(Fila* fi);
11  int consulta_Fila(Fila* fi, struct aluno *al);
12  int insere_Fila(Fila* fi, struct aluno al);
13  int remove_Fila(Fila* fi);
14  int tamanho_Fila(Fila* fi);
15  int Fila_vazia(Fila* fi);
16  int Fila_cheia(Fila* fi);
```

Arquivo FilaEstatica.c

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include "FilaEstatica.h" //inclui os protótipos
04  //Definição do tipo Fila
05  struct fila{
06      int inicio, final, qtd;
07      struct aluno dados[MAX];
08  };
```

Inserção na Fila - Sequencial Estática

- Inserir um elemento em uma **fila sequencial estática** é uma tarefa bastante semelhante à inserção no final de uma **lista sequencial estática**

Inserindo um elemento na fila

```
01  int insere_Fila(Fila* fi, struct aluno al){
02      if(fi == NULL)
03          return 0;
04      if(fi->qtd == MAX)
05          return 0;
06      fi->dados[fi->final] = al;
07      fi->final = (fi->final+1)%MAX;
08      fi->qtd++;
09      return 1;
10  }
```

Inserção na Fila - Sequencial Estática

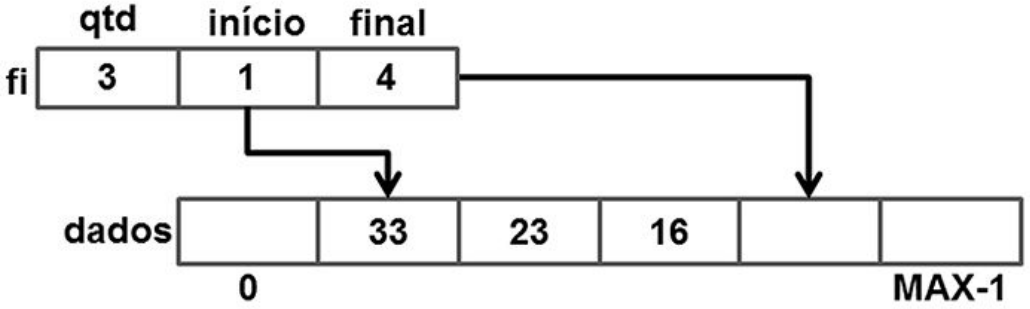
- Primeiramente, verificamos se a Fila **fi** passada não é válida (igual a **NULL** na linha 2).
- Se a fila foi criada com sucesso, precisamos verificar se ela não está cheia (linhas 4 e 5).
- A inserção em uma fila ocorre sempre no seu final. Sendo assim, copiamos os dados a ser inseridos para a posição apontada pelo campo **final** da fila (linha 6).
- Em seguida, devemos incrementar o valor do campo final, indicando assim a próxima posição vaga na fila (linha 7).

Inserindo um elemento na fila

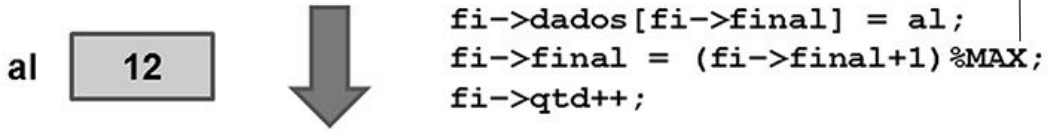
```
01  int insere_Fila(Fila* fi, struct aluno al){
02      if(fi == NULL)
03          return 0;
04      if(fi->qtd == MAX)
05          return 0;
06      fi->dados[fi->final] = al;
07      fi->final = (fi->final+1)%MAX;
08      fi->qtd++;
09      return 1;
10  }
```

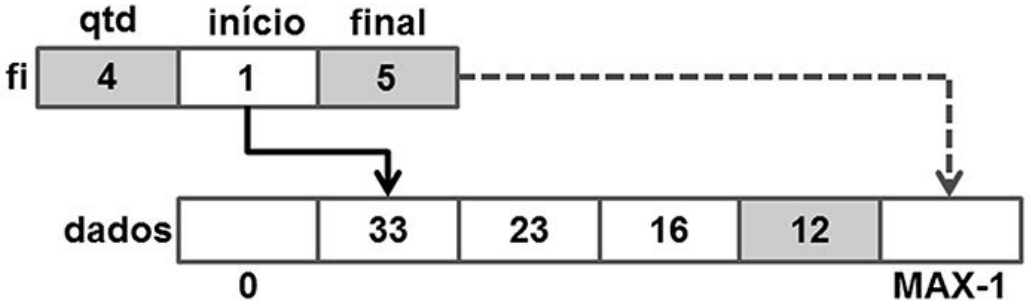
* Note que utilizamos a operação de resto da divisão no cálculo do novo final da fila. Fazemos isso para simular uma fila circular. Assim, ao chegar à posição **MAX** (que não existe no array) o final da fila será colocado na posição **ZERO**, de modo que as posições no começo do array que ficarem vagas, à medida que inserimos e removemos elementos da fila, poderão ser usadas pelo final da fila.

Inserção na Fila - Sequencial Estática

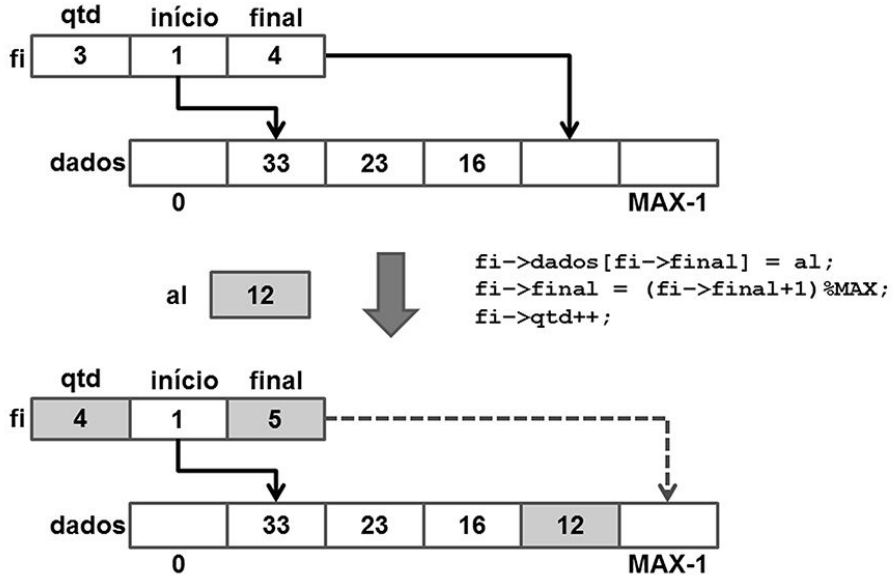


MAX=6
fi->final+1=5
5%6=5





Inserção na Fila - Sequencial Estática



Continue o exemplo adicionando mais dois elementos ao final da fila:

- o elemento al=13 e
- o elemento al=14

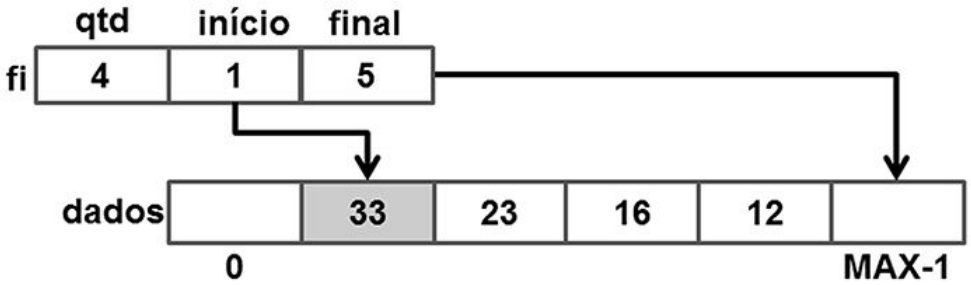
Remoção na Fila - Sequencial Estática

- Remover um elemento de uma **fila sequencial estática** é uma tarefa bastante semelhante à remoção do final de uma **lista sequencial estática**.
- As condições garantem que temos uma fila válida para trabalhar (ou seja, não houve problemas na criação da fila) e que existem elementos que podem ser removidos da fila. Assim, optamos por retornar o valor **ZERO** para indicar que uma das condições é falsa (linha 3).
- Como a remoção é feita no início da fila, basta incrementar em uma unidade o seu valor (linha 4).

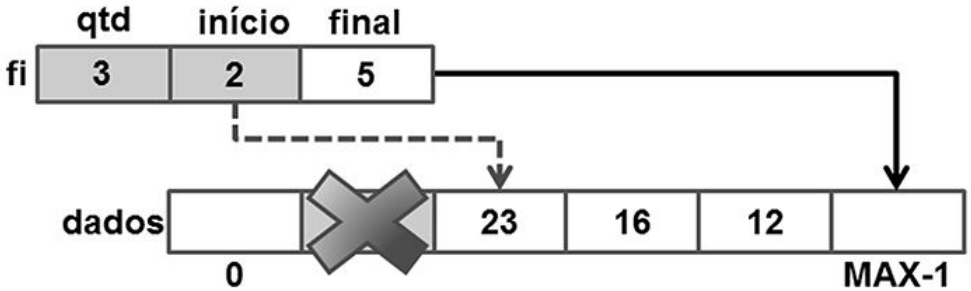
Removendo um elemento da fila

```
01  int remove_Fila(Fila* fi){
02      if(fi == NULL || fi->qtd == 0)
03          return 0;
04      fi->inicio = (fi->inicio+1)%MAX;
05      fi->qtd--;
06      return 1;
07  }
```

Remoção na Fila - Sequencial Estática




```
fi->inicio = (fi->inicio+1)%MAX;
fi->qtd--;
```



Consultando a Fila - Sequencial Estática

Consultando a fila

```
01  int consulta_Fila(Fila* fi, struct aluno *al){
02      if(fi == NULL || fi->qtd == 0
03          return 0;
04      *al = fi->dados[fi->inicio];
05      return 1;
06  }
```

FILA DINÂMICA ENCADEADA

- Uma **fila dinâmica encadeada** é uma fila definida utilizando alocação dinâmica e acesso encadeado dos elementos.
- Armazenamos **o início**, o **final** e a **quantidade** de elementos (**dados**) inseridos na fila.
- Muito Parecido com a LISTA DINÂMICA ENCADEADA!
  - Cada elemento da fila é alocado dinamicamente, à medida que os dados são inseridos dentro da fila, e tem sua memória liberada, à medida que é removido.
  - Esse elemento nada mais é do que um ponteiro para uma estrutura contendo dois campos de informação o dado e o ponteiro para o próximo elemento.
  - A principal vantagem de se utilizar uma abordagem dinâmica e encadeada na definição da fila é a melhor utilização dos recursos de memória, não sendo mais necessário definir previamente o tamanho da fila. Já a sua principal desvantagem é a necessidade de percorrer toda a fila para destruí-la.
    - o ideal é utilizar uma **fila dinâmica encadeada** quando não há necessidade de garantir um espaço mínimo para a execução da aplicação, ou quando o tamanho máximo da fila não é bem definido.

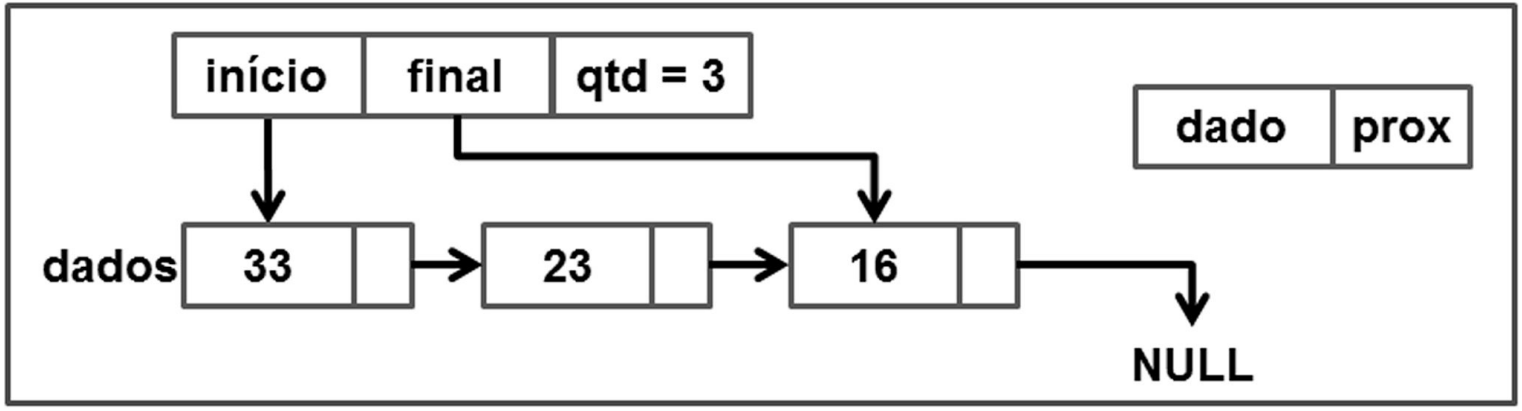
FILA DINÂMICA ENCADEADA - Implementação

Elemento nada mais é do que uma estrutura contendo dois campos: dado e prox

A **fila** nada mais é do que uma estrutura contendo três campos:

- Um ponteiro **inicio**, que indica o primeiro elemento da fila.
- Um ponteiro **final**, que indica o último elemento da fila.
- Um campo **qtd** do tipo **int**, que armazena o número de elementos dentro da fila.

```
Fila *fi;
```



FILA DINÂMICA ENCADEADA - Implementação

Arquivo ListaDinEncad.h

```
01  struct aluno{
02      int matricula;
03      char nome[30];
04      float n1,n2,n3;
05  };
06  typedef struct elemento* Lista;
07
08  Lista* cria_lista();
09  void libera_lista(Lista* li);
10  int insere_lista_final(Lista* li, struct aluno al);
11  int insere_lista_inicio(Lista* li, struct aluno al);
12  int insere_lista_ordenada(Lista* li, struct aluno al);
13  int remove_lista(Lista* li, int mat);
14  int remove_lista_inicio(Lista* li);
15  int remove_lista_final(Lista* li);
16  int tamanho_lista(Lista* li);
17  int lista_vazia(Lista* li);
18  int lista_cheia(Lista* li);
19  int busca_lista_mat(Lista* li, int mat, struct aluno *al);
20  int busca_lista_pos(Lista* li, int pos, struct aluno *al);
```

Arquivo ListaDinEncad.c

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include "ListaDinEncad.h" //inclui os protótipos
04  //Definição do tipo lista
05  struct elemento{
06      struct aluno dados;
07      struct elemento *prox;
08  };
09  typedef struct elemento Elem;
```

Arquivo FilaDin.h

```
01  struct aluno{
02      int matricula;
03      char nome[30];
04      float n1,n2,n3;
05  };
06  typedef struct fila Fila;
07
08  Fila* cria_Fila();
09  void libera_Fila(Fila* fi);
10  int consulta_Fila(Fila* fi, struct aluno *al);
11  int insere_Fila(Fila* fi, struct aluno al);
12  int remove_Fila(Fila* fi);
13  int tamanho_Fila(Fila* fi);
14  int Fila_vazia(Fila* fi);
15  int Fila_cheia(Fila* fi);
```

Arquivo FilaDin.c

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include "FilaDin.h" //inclui os Protótipos
04  //Definição do tipo Fila
05  struct elemento{
06      struct aluno dados;
07      struct elemento *prox;
08  };
09  typedef struct elemento Elem;
10  //Definição do Nó Descritor da Fila
11  struct fila{
12      struct elemento *inicio;
13      struct elemento *final;
14      int qtd;
15  };
```

Exercícios

1) Implemente uma função que cria uma nova fila, faz uma cópia da fila de entrada libera a fila original, seguindo a seguinte declaração da função:
   Fila* copialibera_Fila(Fila* fi);

*Para esse exercício, utilize as funções cria_Fila, consulta_Fila, insere_Fila, remove_Fila e libera_Fila para construir a função copialibera_Fila.

Sua função deve executar em ambos tipos de fila:

- estática : https://www.facom.ufu.br/~backes/wordpress/FilaEstatica.zip
- dinâmica: https://www.facom.ufu.br/~backes/wordpress/FilaDinamica.zip

Exercícios

2) Implemente uma função que receba uma fila e a inverta, de forma eficiente (isso é, passando por cada elemento apenas uma vez), seguindo a seguinte declaração de função:
int inverte_Fila(Fila* fi);

Faça a função para ambos os tipos de fila: **estática** e **dinâmica**. A sua função não deve chamar nenhuma outra função da implementação e alterar a própria fila de entrada (sem criar uma nova fila)

- Para fila dinâmica, sua função deve modificar todos os ponteiros (ant e prox) e trocar o início e fim. O código não deve utilizar filas adicionais.
- Para fila estática, sua função deve trocar a ordem de todos os elementos, trocar início e o final. O código não deve utilizar vetores adicionais.

Referências

Estrutura de Dados descomplicada em Linguagem C (André Backes): Cap 6 ;

Projeto de Algoritmos (Nivio Ziviani): Capítulo 3;

Vídeo aulas (31-37):
https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/

Implementações:

http://www.facom.ufu.br/~backes/wordpress/